

An RM-ODP Based Ontology and a CAD Tool for Modeling Hierarchical Systems in Enterprise Architecture

Lam-Son Lê, Alain Wegmann

Ecole Polytechnique Fédérale de Lausanne (EPFL)

School of Computer and Communication Sciences

CH-1015 Lausanne, Switzerland

{LamSon.Le, Alain.Wegmann}@epfl.ch

Abstract

Enterprise Architecture (EA) requires modeling enterprises across multiple levels (from markets down to IT systems) – i.e. modeling hierarchical systems. Our goal is to build a Computer Aided Design (CAD) tool for EA. To be able to build this CAD tool, we need an ontology that can be used to describe hierarchical systems. The Reference Model of Open Distributed Processing (RM-ODP) was originally defined for describing IT systems and their environment. RM-ODP can also be suited to general, hierarchical, system modeling and, hence, can be used to model enterprises. In this paper, we first give an overview of our CAD tool and we present then how Part 2 and Part 3 of RM-ODP were integrated to define a computer-interpretable ontology that is used in the CAD tool. This ontology is formalized using the Alloy declarative language. Last, we illustrate how the CAD tool can render Unified Modeling Language (UML) diagrams by showing selected aspects of the hierarchical systems.

Keywords: RM-ODP, Enterprise Architecture, Ontology, Formalization, CAD Tool.

1. Introduction

The goal of enterprise architecture [1] is to align the business systems and the IT systems in order to improve an enterprise's competitiveness. Enterprise architecture (EA) deals with systems perceived as hierarchical. They typically span from business entities (market, company, department...) down to IT entities (e.g. applications, applets, servlets, J2EE beans, COM...). Our goal is the development of an EA design method called SEAM (Systemic Enterprise Architecture Methodology) [1] and of the corresponding tools. SEAM is based on Reference Model of Open Distributed Processing (RM-ODP) [2] and is using a UML-like notation. In this paper, we present how the SEAM modeling ontology was developed by combining RM-ODP Part 2 (i.e.

Foundations) and Part 3 (i.e. Architecture) and by formalizing the result of this combination in Alloy [3]. The originality of this paper is in the application of RM-ODP in the context of EA, in the combination of Part 2 and Part 3 and in the realization of a CAD tool. The snapshots used in the paper are made with our tool that implements the SEAM ontology. At the end of the paper, we present how UML-like diagrams can be generated from models developed with this ontology.

The "Reference Model - Open Distributed Processing" (RM-ODP) [2] is an ISO/ITU standard approved in 1996. It provides the definitions and relations between concepts useful to describe object-oriented distributed systems. It positions itself as a "meta-standard" for object-oriented modeling standards. The Object Management Group community adopted in 1998 this standard as a base for describing CORBA systems. RM-ODP has four parts: Part 1 is non-normative and introduces the standard. Part 2 defines the foundations. Part 3 describes the viewpoints necessary to design an IT system. Part 4 is a formalization of the RM-ODP concepts. RM-ODP defines the concepts in a narrative way.

To be able to build a Computer Aided Design (CAD)¹ tool based on RM-ODP particularly for modeling hierarchical systems in EA, we need to have a formal description of the terms defined in RM-ODP. This work was done for RM-ODP Part 2, using the specification language Alloy [3] and the results were published in [4] and [5]. However, even if the terms in Part 2 are formalized in Alloy, this is not sufficient to build a CAD tool. In EA, the concept of hierarchical system modeling is crucial (hierarchical models allow making simpler representations of complicated systems). To be able to build a CAD tool for EA, it is important to specify how the terms defined in Part 2 can be used to represent hierarchical systems. This is why we need to combine

¹ We call a visual modeling tool for hierarchical systems in EA a CAD tool because its scope is mainly towards marketing and business process modeling rather than software modeling (even if software modeling is possible).

them with Part 3 that defines the modeling viewpoints. Once the terms in Part 2 and Part 3 have been combined and formalized, it is possible to build a CAD tool based on this formalization. Then, the tool can generate UML-like diagrams by filtering the elements to be shown.

Section 2 discusses the integration of RM-ODP Part 2 and Part 3. It illustrates our tool capabilities. Section 3 gives the formalization of our ontology in Alloy. Section 4 presents how UML-like diagrams can be generated from the model defined using our ontology. Section 5 outlines the related work.

2. Model Elements Defined by Integrating Viewpoints and Modeling Concepts of RM-ODP

In this section, we explain which viewpoints are necessary to make EA models (Section 2.1) and which model elements are defined in the different viewpoints (Section 2.2). Section 2.1 is based on RM-ODP Part 3 (i.e. Architecture) and Section 2.2 is based on RM-ODP Part 2 (i.e. Foundations).

2.1. Viewpoints and Model Elements in EA

EA has requirements that are different from regular IT system analysis and design. In EA, the development teams deal with hierarchical systems that spans from business down to IT. For example, in an EA project implementing a new way to manage price updates in a nation-wide department store required the development of an enterprise with ten hierarchical levels (from market down to Java programming classes). In a regular IT system specification, fewer levels are considered and are usually not represented in a systematic way. In EA, the teams work with an enterprise model that represents all relevant aspects of the company. Our goal is to define a way to build such hierarchical enterprise models that represent all levels in a systematic way. Thanks to this, the different specialists, who are in charge of the different levels, can share a common way to reason about the enterprise model (because all levels represent systems) while having level-specific heuristics to guide their design decision (an organization with people is essentially not designed in a same way as a software is).

Our goal is model “general systems” (regardless of the fact that they are IT related or business related). The main characteristic of the systems is that they can be considered as wholes or as composite. Systems represented as whole exhibit emergent properties. Systems represented as composite expose their construction. In RM-ODP Part 3, the information viewpoints are defined to describe systems considered as atomic (or as wholes); computational viewpoints are

defined to describe non-atomic systems (or composite systems). In an information viewpoint, there is an information specification describing the system properties in terms of information objects. In a computational viewpoint, there is a computational specification describing the system’s construction in terms of component computational objects. So, in SEAM, we consider the computational specification and the information specification as very important descriptions of the systems.

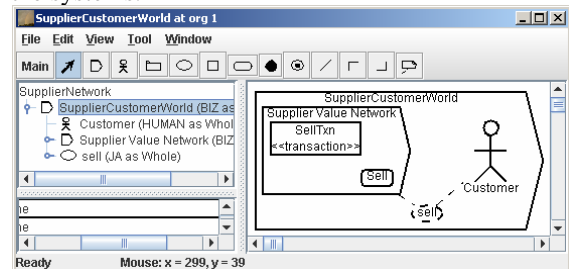


Figure 1. Information specification of the computational object “supplier value network”

As we model hierarchical system, we believe that it is important to make explicit which system is described by these specifications, so we always consider that we have information specifications of computational objects and computational specifications of computational objects. Let’s illustrate this by an example. We model a group of companies collaborating to serve a customer. We call this group of companies a supplier value network. The supplier value network can be considered as a computational object. The information specification of this computational object describes the customer perception of this supplier value network. Figure 1 is a snapshot of our CAD tool [6] that shows the information specification of the supplier value network considered as a computational object. “SellTxn” and “Sell” are the main information object and the main localized action of the supplier, respectively.

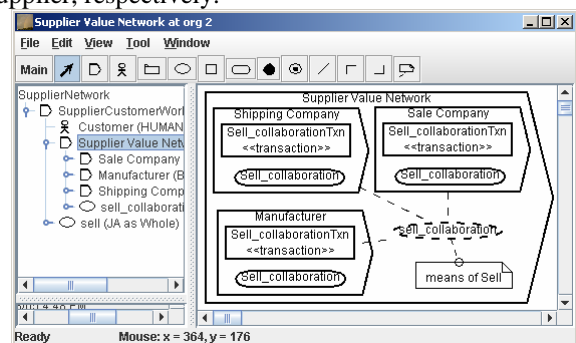


Figure 2. Computational specification of the computational object “supplier value network”

The computational specification of this computational object describes which company collaborates to serve the

customer is illustrated in Figure 2. It is a snapshot showing 3 companies collaborating within the supplier. Note that, in the computational specification of the supplier, each company is described in terms of its information specification.

In summary our EA models are composed of a series of levels, called “organizational levels”, in which computational objects are represented. These computational objects can be either specified by their information specifications or by their computational specifications.

RM-ODP defines five viewpoints to represent systems; enterprise, information, computational, engineering and technology viewpoints. We have discussed our use of the information and computational viewpoints which are central in SEAM. We analyze now the other viewpoints.

In SEAM we do not use the enterprise viewpoints. When designing an IT system, it is important to represent the environment in which the IT system exists. In SEAM, this is done through modeling all the organizational levels which are above the IT system: e.g. department level, company level or market level. In an enterprise viewpoint, all these levels are “collapsed” in the enterprise viewpoint. In SEAM, we keep all these levels separate. Note that some of the concepts defined in the enterprise language are still used in our modeling approach. The only real difference with the enterprise language is that we keep the organizational levels separate [7].

The engineering viewpoints and the technology viewpoints can be found in each organizational level. They are used to define the behavior templates that will have to be implemented in each level. For example, in an EA model, in the organizational level that represent people, the engineering viewpoint and the technology viewpoint are used to generate job descriptions. In the organizational level that describes Java programs, the engineering and technology viewpoints are used to generate the Java source code. The detailed discussion of the engineering viewpoints and of the technology viewpoints are outside the scope of this paper. Related information can be found in [8].

In RM-ODP Part 3, it is stated that each viewpoint is composed of model elements that are defined in RM-ODP Part 2.

An information specification (of a computational object) is defined as made of information objects. In SEAM we consider that these information objects are modified by actions that represent the computational object’s behavior. In SEAM, we call these actions localized actions – a term borrowed from Catalysis [9]. For instance, in the example of a supplier value network in Figure 1, the supplier value network information specification represents the products and the customer

information as information objects and the exchanges with the customer as a localized action.

A computational specification (of a computational object) is defined as made of computational objects that participate to actions. In SEAM, we call these actions, joint actions; a term also borrowed from Catalysis. For example, in the supplier value network, the computational specification represents the companies (i.e. computational objects described by their information specifications) participating to joint actions.

Information specifications can be described at different levels of details. This is useful to represent the goal of the systems (typically represented as localized actions or joint actions seen as a whole) or the way the goal is achieved (typically represented as localized actions or joint actions seen as a composite). For instance, the supplier value network has a “Sell” localized action in Figure 2 that represents, as a whole, its capability to sell books. This action can be broken down into “Get Order”, “Deliver Book” and “Get Payment” as shown in Figure 3. This composite localized action represents a means to achieve the goal. In SEAM, we call “functional levels” this capability to describe information specifications at different levels of details.

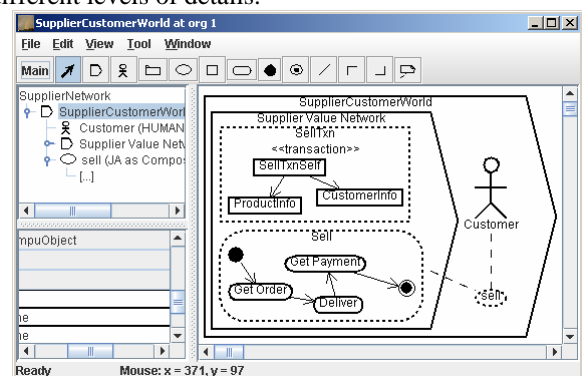


Figure 3. Information specification of computational object “supplier value network” seen at a lower functional level than that in Figure 2

In summary, to define an EA model, it is necessary to have organizational levels that describe the construction of the hierarchical systems (e.g. value networks made of companies, companies made of departments, departments made of IT systems and people, IT system made of Java programs...). In each organizational level, we have computational objects that represent systems and joint actions between them. These computational objects can be described with their information specifications (defined as localized actions and information specifications) or computational specifications (defined as computational objects and joint actions). The information specification of a computational object describes the object as a black-box. Its computational specification describes its construction.

The difference between both is substantial as both descriptions correspond to different levels of reality. For example, the computational specification of an enterprise might include departments and the concept of department would likely not appear at all in its information specification. So, the difference between both is not a simple behavioral refinement. The information specifications can be defined at different functional levels. The definition of these concepts is published in [10]. A concrete and real application of our approach is detailed in [11]. The problem now consists in understanding how these terms can be defined from RM-ODP Part 2.

2.2. Definition of Model Elements

According to RM-ODP Part 2, an entity is any concrete or abstract thing of interest in the universe of discourse. An entity is represented in the model as a model element. Each model element can be characterized by a basic modeling concept (BMC) and by a specification concept (SC) [12]. Examples of basic modeling elements are: object, action, state. Examples of specification concepts are: composite object, type, and instance.

The main concept is the computational object. A computational object is a kind of object as defined in RM-ODP Part 2. As stated in Part 2, an object has states and actions.

- The state of the computational object is described by information objects that have states. The behavior is described by actions that modify information objects. We call this kind of action a localized action. The localized actions are defined in terms of pre-conditions and post-conditions that change the state of the information objects (the dynamic schema defined in Part 3). Additional specification can be captured such as static schemas and invariant schemas that represent the states of the information objects at a given time or at all time. This description corresponds to the information specification of a computational object.
- The computational objects interact with other computational objects. So, we have another kind of actions that represent the collaboration of a computational object with the other computational objects in its environment. We call this action a joint action. The joint actions change the state of the information objects of the computational objects participating in the collaboration.

In summary, to build an EA model, we have the following basic modeling concepts: computational objects, information objects, localized actions, joint

actions and states. RM-ODP defines all the concepts even if it does not name them explicitly (for example, the joint action is not defined but actions can involve more than one object). Our contribution to RM-ODP was to name some of these already existing concepts explicitly.

As we should be able to describe different functional levels, all information objects, joint actions and localized actions can be represented as whole or as composite. Of course, to be able to represent organizational levels, the computational objects can be represented as whole or as composite as well. This means that we need to have computational objects as whole or as composite, information objects as wholes or as composite, joint actions as wholes or as composite, localized actions as whole or as composite.

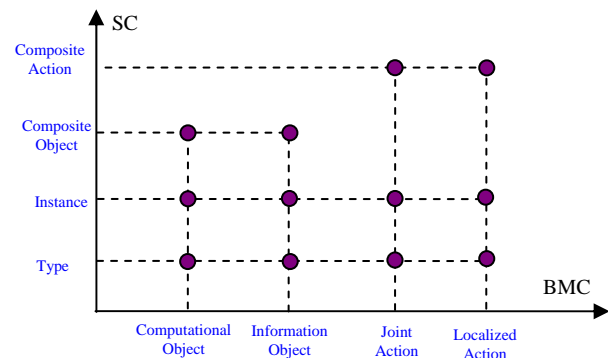


Figure 4. The 16 modeling elements can be defined by combining necessary BMCs and SCs

RM-ODP defines model elements by combining a basic modeling concept (such as object or action) and a specification concept (such as type or instance). To generate our ontology, instead of combining generic objects and actions with type and instance, we combine computational objects, information objects, localized action and joint actions with type and instance (as illustrated in Figure 4). In addition, we need to add if the model element is considered as a whole or as a composite. In summary, our model elements are defined by the following statements:

- $\langle x \rangle$ type as a whole
- $\langle x \rangle$ type as composite
- $\langle x \rangle$ instance as a whole
- $\langle x \rangle$ instance as composite

Where $\langle x \rangle$ is computational object, information object, localized action or joint action.

3. Formalization

Although we consider that $\langle x \rangle$ seen as whole and $\langle x \rangle$ seen as composite are two separate model elements, we notice that these two model elements always go together. In fact, they represent a single entity in reality.

In the EA model, depending on the context, $\langle x \rangle$ seen as whole or $\langle x \rangle$ seen as composite is used. To simplify the formalization, when we define $\langle x \rangle$, we always consider both aspects (whole and composite).

The formalization is done in Alloy 2.0 [3]. Alloy is a lightweight modeling language based on relations and takes the syntax of the first-order logic. Since version 2.0, the language also has object-oriented constructs. The language is accompanied by a tool called Alloy Constraint Analyzer (ACA) that can be used for simulating and checking models written in Alloy.

The entire Alloy code is given in the appendix at the end of this paper. Note that the Alloy code formalizes the types ($\langle x \rangle$ type), so all terms in the Alloy could be prefixed with “Type”. For clarity reasons, we decided not to do so.

Table 1 shows signature `CompuObject` (sig is a keyword in Alloy) that formalizes the computational object. Each computational object optionally mediates a joint action (field `ja`), refers to an information object (field `info`) and a localized action (field `la`), has a number of child computational objects (field `compu_children`) and optionally has a parent computational object (field `compu_parent`). We can interpret that the field `ja` and the field `compu_children` correspond to a computational object seen as composite, whereas the field `compu_parent`, the field `info` and the field `la` stands for that computational object but seen as whole. In other words, the computational object as whole and as composite are formalized in one Alloy signature: `CompuObject`.

Table 1. Alloy code for the computational object

```
sig CompuObject // definition of the computational object
{ // declaration of all fields of the computational object
  ja : option JointAction,
  info : option InfoObject,
  la : option LocalizedAction,
  compu_children : set CompuObject,
  compu_parent : option CompuObject
} { // invariants concerning the computational object
  #compu_parent > 0 => this in compu_parent::compu_children
  all c : compu_children | c::compu_parent = this
  #ja > 0 => ja::compu_med = this
  all j : JointAction - ja | j::compu_med = this =>
    j in ja.^joint_children
  #info > 0 => info::compu_host = this
  all io : InfoObject - info | io::compu_host = this =>
    io in info.^info_children
  #la > 0 => la::compu_owner = this
  all l : LocalizedAction - la | l::compu_owner = this =>
    l in la.^localized_children
}
```

The second block between curly brackets, right after the declaration of all fields of signature `CompuObject`, lists all invariants that must be held for any computational objects. The first line of this block assures that any computational object must be a child computational object of its parent. The second line implies that all child computational objects have the same parent. The rest of this block concerns the way each computational object relates to information objects, joint

actions and localized actions. In short, these invariants mandate the way model elements are related to one another in our ontology.

In EA models, having individual model elements is not enough. Indeed, we need to put them in relation. First, each model element has child elements which are of the same kind as the parent element. Second, to express static schema in the information specification, information objects need to be related to other information objects by associations. Third, to express the activity of each computational object, its localized actions need to be related by action transitions. Fourth, to correctly show the interaction between computational objects, links between computational objects and joint actions they participate in must be kept. As a summary, the following relations need to be addressed in the ontology

- parent-child relation
- association
- action transition
- collaboration link

One way to formalize these relations is to declare additional fields in the signatures that formalize model elements. This first approach applies to the child-parent relation. For example, in signature `CompuObject` we can put a field such as `compu_children : set CompuObject` that results in all child computational objects of the declared computational object. Another approach is to declare a particular signature dedicated for each kind of relation. This signature must have at least two fields referencing the source and the destination model element of the declared relation. In this approach, it is easy to put additional attributes such as role name, cardinality... to declared relations. The second approach applies to association, action transition and collaboration link. Their corresponding signatures in Alloy code are named `IO2IO`, `LA2LA`, `CO2JA` and `JA2CO` respectively. This naming convention is put in place to emphasize which kinds of model elements are connected by the formalized relations.

The entire formalization code is simulated using ACA at the size of 4 (i.e. at most 4 instances shall be created for each signature). Figure 5 shows one possible solution. The signature instances are represented as eclipses. For each instance, the fields are drawn as outgoing arrows. The eclipse that an arrow points to represents the value of the field represented by that arrow. Note that to fully formalize the ontology, we also need to declare attributes in the signatures of model elements such as pre/post condition (of the localized action and the joint action), role name or cardinality (of the association and the collaboration link). These declarations will lead to defining special Alloy signatures to represent primitive data types (e.g. Integer, String...) and will make the simulation unnecessarily complicated.

For this reason, we omit all primitive attributes and only declare the ones that represent child-parent relation.

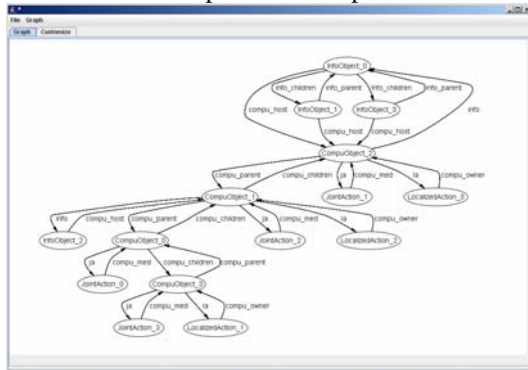


Figure 5. A simulation of the formalization written in Alloy

Table 2 list some correspondences between the model of the supplier value network shown in Figure 1, 2, 3 and the Alloy solution shown in Figure 5.

Table 2. Comparison between a model done in our CAD tool and the solution found by ACA

Model Element	Model in CAD tool	Solution by ACA
Computational Object	Supplier Customer World	CompuObject_0
Computational Object	Supplier Value Network	CompuObject_1
Computational Object	Customer	CompuObject_3
Computational Object	Sale Company	CompuObject_2
Joint Action	sell	JointAction_0
Information Object	SellTxn	InfoObject_2
Localized Action	Sell	LocalizedAction_2
Joint Action	sell collaboration	JointAction_2

The design of the CAD tool follows Model-View-Controller approach. Each modeling concept declared in Alloy corresponds to a design class in the part “Model” and is rendered by, nevertheless, two classes in the part “View”: one class for $\langle x \rangle$ as whole and the other for $\langle x \rangle$ as composite.

4. Example of UML Representation

The information specification of the Supplier Value Network can be filtered to show only the class diagram (Figure 6 a) or the activity diagram (Figure 6 b). The modeler can easily do so while selecting the pictogram of the Supplier Value Network in the diagram illustrated in Figure 3.

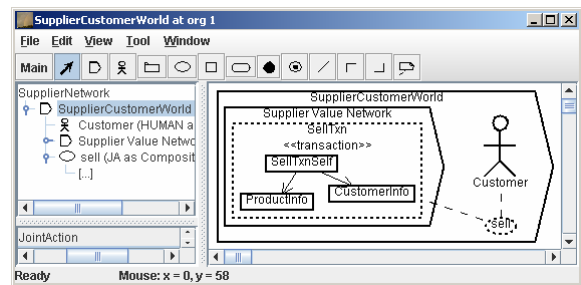


Figure 6 a) Filtered information specification shows a class diagram of “supplier value network” in its context.

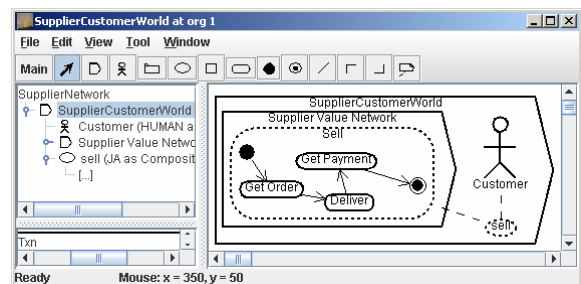


Figure 6 b) Filtered information specification shows an activity diagram of “supplier value network” in its context.

The value of this representation lies in the fact that the modeler can ultimately reason about each system by using individual UML diagrams. Future versions of our tool will allow her to edit these diagrams in separate windows, making it more UML-compatible.

5. Related Work

RM-ODP based modeling has increasingly attracted researchers in the field of EA and formal methods. [13] proposes a tool environment for viewpoint-oriented EA. This framework allows the modeler to define various viewpoints (that are not necessarily limited to RM-ODP viewpoints) of an EA project. [14] aims at relating RM-ODP enterprise viewpoint and computational viewpoint to make consistency checking feasible. [15] explores the possibility of formally writing computational viewpoint specifications in Maude, an object-oriented executable rewriting logic language. [16] proposes a declarative language (with specialized constructs for security) to specify enterprise viewpoints. [17] describes a formal interpretation to viewpoint consistency and proposes some strategies for checking viewpoint consistency in general. [18] focuses on the consistency between the computational viewpoint and the engineering viewpoint. Our work addresses hierarchical systems in EA and tries to integrate the information viewpoint and the computational viewpoint of RM-ODP to make a

computer-interpretable tool-supported ontology with quite precise semantics of model elements. We use Alloy, a lightweight modeling language based on set theory with object-oriented constructs, to formalize our ontology.

One of the big concerns about RM-ODP viewpoints is the transition between them. [19] defines information actions and proposes the articulation between them and messages in computational view. [20] establishes one-to-one mapping from an information action to a stateless object. In our ontology, information objects representing the transactions are similar to stateless objects. We believe that, for a certain computational object, its computational specification is quite independent from its information specification. The decomposition made on a computational specification is just a choice of the modeler, although it can be inspired by the information specification. Our tool manages the connection between these two specifications. However, it leaves the decomposition to the modeler as her own decision.

Conclusions

In this paper, we present an ontology that is suited to modeling hierarchical systems in EA. We base our work on RM-ODP. To define the necessary model elements and their relations in the ontology, we integrate the computational viewpoint and the information viewpoint of RM-ODP. The resulting ontology is formalized in Alloy and simulated using Alloy Constraint Analyzer. The computer-interpretable version of the ontology is implemented in a CAD tool. This tool allows the modeler to build models for hierarchical systems where every model element can be seen either as whole or as composite. This tool can also render UML diagrams according to some filtering option to reflect particular aspects of the system of interest.

Our future work investigates the possibility to translate the formalization written in Alloy to partial Java code (mainly in the "Model" part of the entire Model-View-Controller design of the tool) that implements our CAD tool so that we can achieve some automation between the ontology specification and the ontology implementation. We also consider developing advanced features that allows the modeler to separately edit UML diagrams of the system of interest.

References

[1] Wegmann, A., "On the Systemic Enterprise Architecture Methodology (SEAM)," presented at 5th ICEIS, Angers, France, April 2003.

[2] OMG, "ISO/IEC 10746-1, 2, 3, 4 | ITU-T Recommendation, X.901, X.902, X.903, X.904, Reference Model of Open Distributed Processing," 1995-1996.

[3] MIT, The Alloy Constraint Analyzer, <http://alloy.mit.edu/>

[4] Naumenko, A., Wegmann, A., Genilloud, G., and Frank, W. F., "Proposal for a formal foundation of RM-ODP concepts," presented at WOODPECKER / 3rd ICEIS workshop, Setúbal, Portugal, July 2001.

[5] Naumenko, A. and Wegmann, A., "A Metamodel for the Unified Modeling Language," presented at <<UML>> 2002, Dresden, Germany, September/October 2002.

[6] Lê, L. S. and Wegmann, A., "SeamCAD 1.x: User's Guide," School of Computer and Communication Sciences, EPFL, Lausanne IC/2004/98, November 2004.

[7] Tyndale-Biscoe, S., "RM-ODP Enterprise Language ITU-T Rec.X.911: ISO/IEC 15414," 2005.

[8] Wegmann, A. and Preiss, O., "MDA in Enterprise Architecture? The Living SystemTheory to the Rescue...," presented at 7th IEEE EDOC, Brisbane, Australia, September 2003.

[9] D'souza, D. F. and Wills, A. C., *Object, Components and Frameworks with UML, The Catalysis Approach*: Addison-Wesley, 1999.

[10] Lê, L. S. and Wegmann, A., "Definition of an Object-Oriented Modeling Language for Enterprise Architecture," presented at 38th Hawaii International Conference on System Sciences, Hawaii, USA, January 2005.

[11] Wegmann, A., Regev, G., and Loison, B., "Business and IT Alignment with SEAM," presented at REBNITA / 13th IEEE RE workshop, Paris, September 2005.

[12] Wegmann, A. and Naumenko, A., "Conceptual Modeling of Complex Systems using an RM-ODP based Ontology," presented at 5th IEEE EDOC, Seattle, USA, September 2001.

[13] Steen, M. W. A., Akehurst, D. H., Doest, H. W. L., and Lankhorst, M. M., "Supporting Viewpoint-Oriented Enterprise Architecture," presented at 8th IEEE EDOC, California, USA, September 2004.

[14] Dijkman, R., Quartel, D., Pires, L., and Sinderen, M., "A Rigorous Approach to Relate Enterprise and Computational Viewpoints," presented at 8th IEEE EDOC, California, USA, September 2004.

[15] Romero, R. and Vallecillo, A., "Formalizing ODP Computational Viewpoint Specifications in Maude," presented at 8th IEEE EDOC, California, USA, September 2004.

[16] Lupu, E., Sloman, M., Dulay, N., and Damianou, N., "Ponder: realising enterprise viewpoint concepts," presented at 4th IEEE EDOC, Makuhari, Japan, September 2000.

[17] Bowman, H., Steen, M., Boiten, E., and Derrick, J., "A Formal Framework for Viewpoint Consistency," *Formal Methods in System Design*, 2002.

[18] Boiten, E., Bowman, H., Derrick, J., Linington, P., and Steen, M., "Viewpoint consistency in ODP," *ELSEVIER*, 2000.

[19] Dustzadeh, J. and Najm, J., "Consistent Semantics for ODP Information and Computational Models," presented at FORTE / PSTV'97, Osaka, Japan, November 1997.

[20] Bernardeschi, C., Dustzadeh, J., Fantechi, A., Najm, J., Nimour, A., and Olsen, F., "Transformation and Consistent Semantics for ODP Viewpoints," presented at FMOODS'97, Canterbury, UK, July 1997.

Appendix: Ontology Formalization in Alloy

```

module seamcad

sig CompuObject{
  ja : option JointAction,
  info : option InfoObject,
  la : option LocalizedAction,
  compu_children : set CompuObject,
  compu_parent : option CompuObject
} {
  #compu_parent > 0 => this in compu_parent::compu_children
  all c : compu_children | c::compu_parent = this
  #ja > 0 => ja::compu_med = this
  all j : JointAction - ja | j::compu_med = this => j in ja.^joint_children
  #info > 0 => info::compu_host = this
  all io : InfoObject - info | io::compu_host = this => io in info.^info_children
  #la > 0 => la::compu_owner = this
  all l : LocalizedAction - la | l::compu_owner = this => l in la.^localized_children }

fact compu_acyclic {
  all c : CompuObject | (c !in c.^compu_parent && c !in c.^compu_children) }

sig InfoObject{
  compu_host : CompuObject,
  info_children : set InfoObject,
  info_parent : option InfoObject
} {
  all c : info_children | c::info_parent = this && c::compu_host = this::compu_host
  #info_parent > 0 => this in info_parent::info_children }

fact info_acyclic {
  all c : InfoObject | (c !in c.^info_parent && c !in c.^info_children) }

sig JointAction {
  compu_med : CompuObject,
  joint_children : set JointAction,
  joint_parent : option JointAction
} {
  all c : joint_children | c::joint_parent = this && c::compu_med = this::compu_med
  #joint_parent > 0 => this in joint_parent::joint_children }

fact joint_acyclic {
  all c : JointAction | (c !in c.joint_parent && c !in c.^joint_children) }

sig LocalizedAction {
  compu_owner : CompuObject,
  localized_children : set LocalizedAction,
  localized_parent : option LocalizedAction
} {
  all c : localized_children | c::localized_parent = this && c::compu_owner = this::compu_owner
  #localized_parent > 0 => this in localized_parent::localized_children }

fact localized_acyclic {
  all c : LocalizedAction | (c !in c.^localized_parent && c !in c.^localized_children) }

sig IO2IO { // association
  source : InfoObject,
  destination : InfoObject
} {
  source::compu_host = destination::compu_host }

sig LA2LA { // action transition
  source : LocalizedAction,
  destination : LocalizedAction
} {
  source::localized_parent = destination::localized_parent
  source::compu_owner = destination::compu_owner }

sig CO2JA { // collaboration link
  source : CompuObject,
  destination : JointAction
} {
  source::compu_parent = destination::compu_med }

sig JA2CO { // collaboration link
  source : JointAction,
  destination : CompuObject
} {
  source::compu_med = destination::compu_parent }

sig IO2CO { // trace dependency
  source : InfoObject,
  destination : CompuObject }

fact oneRoot {
  sole co : CompuObject | #co.compu_parent = 0 }

fun hasLifecycle() {
  some co : CompuObject | one co.ja && no co.la && no co.info && #co.compu_children = 2 }

run hasLifecycle for 4

```